OpenCUDA+MPI

A Framework for Heterogeneous GP-GPU Distributed Computing

Kenny Ballou
Nilab Mohammad Mousa
College of Engineering – Department of Computer Science
Boise State University
Alark Joshi, Ph.D

Abstract

The introduction and rise of General Purpose Graphics Computing has significantly impacted parallel and high-performance computing. Even worse, it has brought about even more challenges when it comes to distributed computing (with GPU's). Current solutions target specifics: specific hardware, specific network topology, a specific level of sameness. What if we could ignore specifics? What if we could write a general algorithm to solve a problem and have a job scheduler do the rest? That is the goal of OpenCUDA+MPI. To achieve this goal, we have written a framework that allows a developer/ data scientist to write a general algorithm/ solution without the overhead of worrying about the specifics of the cluster it will run against. We have found, [nothing yet] and, therefore, can conclude [also nothing].

Keywords: Parallel Computing, Distributed Computing, General Purpose Graphics Processing, High-Performance Computing, Scientific Computing

1 Topic

Increasingly, Graphics Processing Units (GPUs) are being used for general purpose computation (General Purpose GPU (GP-GPU)). They have significantly altered the way high-performance computing tasks can be performed today. To accelerate general-purpose scientific applications, computationally intensive portions of the application are passed to GPUs. Once the GPU has completed its task it sends the result back to the Centeral Processing Unit (CPU) where the application code is running. This process can make applications run noticeably faster.

Scientific applications require a large number of floating point number operations, CPUs are not sufficient to carry out such computationally intensive tasks. CPUs are responsible for prioritization and execution of every instruction. Generally, a processor is described by the number of execution cores it owns. Modern Central Processing Unit have eight cores while GPUs have hundreds or more cores. More cores grant GPUs the ability to perform more tasks at the same time.

In computer architecture there are two main ways of processing: serial and parallel. CPUs consist of a small number of cores (microprocessors) that are best at processing serial data. On the other hand, GPUs consist of thousands of cores that are designed for processing parallel data. Given a program, we can run the parallel portions of the code on GPUs while the serial portions run on the CPU. The programmable GPU has evolved into a highly parallel, multithreaded, many core processor with tremendous computational power. Compute Unified Device Architecture (CUDA) is an architecture for utilizing and distributing computational tasks onto a computer's graphics processing unit(s). As a parallel computing platform and programming model, CUDA utilizes the power of GPU to achieve dramatic increases in computing performance [6]. This fact is well illustrated in Figure 1 Floating-Point Operations per Second for the CPU and GPU.

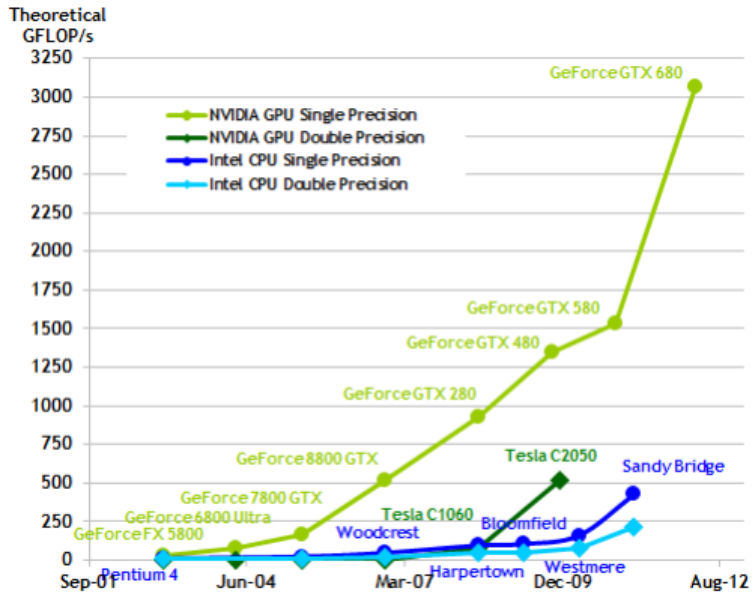

Figure 1: Floating-Point Operations per Second for the CPU and GPU [image source(?)]

Today, more than one million CUDA-enabled GPUs are used by software developers, scientists and researchers in order to obtained a large performance gain in wide-ranging applications [6]. A major challenge remains in being able to seamlessly integrate multiple workstations to further parallelize the computational tasks [15] [17]. Current approaches provide the ability to parallelize tasks, but they are less focused on optimally utilizing the varied capabilities of heterogeneous graphics cards in a cluster of workstations (across many

computer nodes).

We propose to create a framework for using both Message Passing Interface (MPI) and CUDA on a cluster of computers to dynamically and easily assign computationally intensive tasks to the machines participating in the cluster. MPI is a popularly used standardized interface for communication among a group of computers. It facilitates the passing of messages between the nodes in the cluster.

Our framework will be easy to use on a heterogeneous cluster of computers, each containing a CUDA-capable GPU. The framework will optimize the scheduling of low-level computational tasks based on the capabilities of the varied GPUs in the cluster. The framework will provide the system administrator with the ability to easily configure it to allow optimal use of the individual GPUs. Overall, the goal is to create a framework that is simple and easy to use, facilitating the combination of two powerful means of computing to further increase the throughput and overall computing power of a cluster.

We plan to evaluate the efficacy of our framework on the problem of vessel extraction. Currently, we use a single GPU to extract vascular structures from a CT angiography scan which is computationally intensive [14]. Using the framework to accelerate the computational task will provide us with key insights on its usability.

2 Significance

CUDA+MPI will serve to better utilization existing computing resources. In other words, it will give us high compute power at low cost. Many personal computers have a CUDA graphics card. It is more cost effective to build a cluster of multiple computers than to purchase a supercomputer to perform high performance computing. The reason for the low costs stem from the fact that clusters are built of components which are sold in high volumes [14]. CUDA+MPI will have the advantage of handling heterogeneous distributed computing. In other words, CUDA+MPI avoids hardware limitations. All the computers participating the cluster do not need to have the same model of CUDA-enabled GPU cards. This will also give us the freedom to easily add new nodes to the cluster as needed.

Since CUDA+MPI will operate in a heterogeneous cluster it will manage the job requests from the user to be run on a heterogeneous cluster of GPU-capable computers. In order to decide job priority, amount of work and availability of devices will be considered. In addition to improved performance, the proposed framework will grant researchers and scientists the assurance that the heterogeneous cluster will be optimized for computing without requiring extra effort and thought.

3 Related Works

Current approaches to accelerating research computations and engineering applications include parallel computing and distributed computing. A parallel system is when several processing elements work simultaneously to solve a problem faster. The primary consideration is elapsed time, not throughput or sharing of resources at remote locations. A distributed system is a collection of independent computers that appears to its users as a single coherent system. In distributed computing the data is split into smaller parts and subsequently spread across the system. The result is then collected and outputted. Summary of current solutions are listed in Table 1 below followed by detailed description of every entry.

Project	**Description**
Hadoop	Distributed computation framework [7] [4] [21]
BOINC	Volunteer and grid computing project distributed [10]
TORQUE	Job scheduler for distributed computing [9]
GPUDirect	GPU-to-GPU framework for parallel computing [3]
MPI	Message-passing system for parallel computations [24] [11]
PVM	Distributed environment and message passing system [5]

Table 1: Summary of Current Approaches and Projects

3.1 Hadoop

Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm [4] [23] [13] [12]. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts (nodes), and executing application computations in parallel close to their data [21]. The MapReduce [19] [8] paradigm which Hadoop implements is characterized by dividing the application into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. The worker node processes the smaller problem, and passes the answer back to its master node. The answers to the subproblems are then combined to form the output. Hadoop is popular model for distributed computations, however it is does not integrate with CUDA-enabled GPUs.

3.2 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) is an open source middleware system for volunteer and grid computing. The general public can contribute to today's computing power by donating their personal computer's disk space and some percentage of CPU and GPU processing. In other words, BOINC is software that can use the unused CPU and GPU cycles on a computer to do scientific computing [10]. The BOINC project allows for distributed computing using hundreds of millions of personal computers and game consoles belonging to the general public. This paradigm enables previously infeasible research and scientific super-computing. The framework requires that individuals are connected to the internet and is supported by various operating systems, including Microsoft Windows, Mac OS X and various Unix-like systems. Although BOINC is another step in grid computing, it is does not seem to be relevant to our research.

3.3 TORQUE

Terascale Open-Source Resource and QUEue Manager (TORQUE) is a job scheduler, a computer application that controls program execution. It enables users to control computational tasks over distributed compute nodes, as well as scheduling and administration on a cluster [9]. TORQUE has the ability to handle larger clusters with tens of thousands of nodes. It also can handle larger jobs that span hundreds of thousands of processors due to advanced GPGPU scheduling. However, TORQUE does not allow for heterogeneous hardware in the cluster, making it inferior compared to `CUDA+MPI` framework.

3.4 GPUDirect

Currently GPU based clusters are becoming more popular. GPUDirect is a GPU based clustering using infiniband, an architecture specification that defines a connection between processor nodes. GPUDirect allows for multiple GPUs to communicate with each other directly by giving GPUs ability to initiate and manage memory transfer [3]. Prior to GPUDirect, GPU-to-GPU communication involved CPU. GPUDirect performance gain exceed CPU solutions. Direct GPU-to-GPU communication is less computationally expensive since less messages will be sent and received via the host server. An issue with GPUDirect is that it requires a specific hardware. The proposed framework will serve as software solution to avoid hardware limitations.

3.5 MPI

MPI is the dominant message passing programming paradigm for clusters [1] [2]. MPI is a standardized and portable message-passing system that functions on a wide variety of parallel computers. Although the standard MPI defines the syntax and semantics of a core of library routines in the C programming language, it is a language-independent communications protocol used to program parallel computers. `CUDA+MPI` builds upon this model due to the fact that MPI is the dominant model used in high-performance computing [22]. MPI has been implemented for almost every distributed memory architecture and is optimized for the hardware on which it runs.

3.6 PVM

Parallel Virtual Machine (PVM) is a software tool for parallel networking of computers. It is designed to allow a network of heterogeneous Unix and/or Windows machines to be used as a single distributed parallel processor. Thus, large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. PVM enables users to exploit their existing computer hardware to solve much larger problems at less additional cost [5]. PVM was a step towards modern trends in distributed processing and grid computing but has, since the mid-1990s, largely been supplanted by the much more successful MPI standard for message passing on parallel machines.

Proposed `CUDA+MPI` framework will take advantage of both MPI (Message Passing Interface) and CUDA to perform computations on GPU cards of computers participating in the on a cluster. `CUDA+MPI`'s goal is to allow a collection of heterogeneous computers each containing a CUDA-capable GPU to be used as a coherent and flexible concurrent computational resource.

4 Methodology

Our framework will be developed using a number of tools and will also build upon a few already developed and mature software libraries. Overall, we plan to use the Python programming language, MPI for cross process communication, and CUDA for graphics computing.

4.1 Python

We will use Python programming language because of its ease of development and plethora of existing libraries such as MPI. Python also adds some advantages when it comes to usability when needing more performance. For example, if we discover a need for certain aspects of the project to be tuned or otherwise run faster, we can easily switch to C or C++ and write sections of the program in a (more performant) native language.

4.2 MPI

Currently our cluster consists of 16 computers provided to our research lab by the computer science department of Boise State University. We will use MPI because it has established itself as the de-facto interface for cross process communication [24]. To allow for interfacing with Python [11], we will use `mpi4py` because of its maturity and implementation completeness.

4.3 CUDA

Further, we will be using CUDA because of existing knowledge of the framework and also because it is a well established framework for GPU computing.

4.4 `CUDA+MPI`

Our framework's goal is to be able to manage the job queue to be run on GPUs for a cluster of GPU-capable computers. As more implementation work is done one or more of the parameters listed below will be considered to decide which particular job to run.

- Job priority
- Compute resource availability
- Execution time allocated to user
- Number of simultaneous jobs allowed for a node
- Estimated execution time
- Elapsed execution time
- Availability of devices

4.5 Testing

To evaluate the efficacy and accuracy of our framework several tests will be executed. As previously mentioned one of the first algorithms to be tested will be the problem of vessel extraction. Accurately extracting vascular structures from a Computerized Tomographic angiography–also called CT angiography scans–is important for creating oncologic surgery planning tools as well as medical visualization applications [14]. Currently, we use a single GPU to extract vascular structures from a CT angiography scan, which is computationally intensive. The following test programs will be developed as time allows:

- N-Body Simulation
- Prime Number Searching

Every test program will be evaluated in three categories: CPU-only, CUDA-only, and `CUDA+MPI`. Using the framework to accelerate the computational tasks will provide us with key insights on its usability.

A Reference

[1] Mpi: A message passing interface standard. http://www.mpi-forum.org/, June 1995.

[2] Mpi-2: Extensions to the message passing interface. http://www.mpiforum.org/, July 1997.

[3] Gpu-direct accelerating gpu cluster performance. http://www.youtube.com/, May 2010.

[4] Apache hadoop. http://hadoop.apache.org/, April 2013.

[5] Computer science and division - pvm: Parallel virtual machine. http://www.csm.ornl.gov/pvm/, April 2013.

[6] Cuda c programming guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, May 2013.

[7] Hadoop project description. http://wiki.apache.org/hadoop/ProjectDescription, April 2013.

[8] Mapreduce. http://wiki.apache.org/hadoop/MapReduce, April 2013.

[9] Torque resource manager-adaptive computing. http://www.adaptivecomputing.com/products/open-source/torque/, May 2013.

[10] David P Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[11] Lisandro Dalcin. Mpi for python. http://mpi4py.scipy.org/docs/usrman/index.html, Janurary 2012.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters, osdi'04: Sixth symposium on operating system design and implementation, san francisco, ca, december, 2004. *S. Dill, R. Kumar, K. McCurley, S. Rajagopalan, D. Sivakumar, ad A. Tomkins, Self-similarity in the Web, Proc VLDB*, 2004.

[13] Jonas Dias and Albino Aveleda. Hpc environment management: new challenges in the petaflop era. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 293–305. Springer, 2011.

[14] Marius Erdt, Matthias Raspe, and Michael Suehling. Automatic hepatic vessel segmentation using graphics hardware. In *Medical Imaging and Augmented Reality*, pages 403–412. Springer, 2008.

[15] Bilel Hadri, Mark Fahey, and Nick Jones. Identifying software usage at hpc centers with the automatic library tracking database. In *Proceedings of the 2010 TeraGrid Conference*, page 8. ACM, 2010.

[16] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 22–22. USENIX Association, 2011.

[17] Benjamin Hindman, Andy Konwinski, Matei Zaharia, and Ion Stoica. A common substrate for cluster computing. In *Workshop on Hot Topics in Cloud Computing (HotCloud)*, volume 2009, 2009.

[18] C.R. Johnson. Biomedical visual computing: Case studies and challenges. *Computing in science & engineering*, 14(1):12–21, 2012.

[19] Yuan Luo, Zhenhua Guo, Yiming Sun, Beth Plale, Judy Qiu, and Wilfred W Li. A hierarchical framework for cross-domain mapreduce execution. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, pages 15–22. ACM, 2011.

[20] Christopher Parker and Hussein Suleman. A lightweight web interface to grid scheduling systems. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 180–187. ACM, 2008.

[21] Konstantin V Shvachko. Apache hadoop: The scalability update. *login: The Magazine of USENIX*, 36:7–13, 2011.

[22] Sayantan Sur, Matthew J Koop, and Dhabaleswar K Panda. High-performance and scalable mpi over infiniband with reduced memory usage: an in-depth performance analysis. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 105. ACM, 2006.

[23] J. Venner. Pro hadoop. http://www.apress.com/, June 2009.

[24] Wes. A comprehensive mpi tutorial reference. http://www.mpitutorial.com/, 2012.

Glossary

CPU Centeral Processing Unit. 2, 5, 6

CUDA Compute Unified Device Architecture. 2–8

GP-GPU General Purpose GPU. 2

GPU Graphics Processing Unit. 2–7

MPI Message Passing Interface. 3, 6, 7

PVM Parallel Virtual Machine. 6

TORQUE Terascale Open-Source Resource and QUEue Manager. 6